

Designing and Training Feedforward Neural Networks: A Smooth Optimisation Perspective

Hao Shen

E-mail: hao.shen@tum.de

Department of Electrical and Computer Engineering
Technische Universität München, Germany

Abstract

Despite the recent great success of deep neural networks in various applications, designing and training a deep neural network is still among the greatest challenges in the field. In this work, we present a smooth optimisation perspective on designing and training multilayer Feedforward Neural Networks (FNNs) in the supervised learning setting. By characterising the critical point conditions of an FNN based optimisation problem, we identify the conditions to eliminate local optima of the corresponding cost function. Moreover, by studying the Hessian structure of the cost function at the global minima, we develop an approximate Newton FNN algorithm, which is capable of alleviating the vanishing gradient problem. Finally, our results are numerically verified on two classic benchmarks, i.e., the XOR problem and the four region classification problem.

Index Terms

Forward Neural Networks (FNNs), smooth optimisation, critical point analysis, Hessian matrix, approximate Newton's method.

1 Introduction

Recently, Deep Neural Networks (DNNs) have been successfully applied to solve complicated problems in pattern recognition, computer vision, and speech recognition, cf. [1, 2, 3]. Despite its great success, designing and training a deep neural network is still among the greatest challenges in the field, cf. [4]. In this work, we focus on the study of the Feedforward Neural Networks (FNNs). One major reason for the difficulty in designing and training an FNN is that their performance is highly dependent on various factors in a very complicated way. Besides the determinative factor being the architecture of an FNN [5, 6], the work in [7] demonstrates the impact of different activation functions to the performance of an FNN. Moreover, the choice of error functions is also shown to be influential, cf. [8].

Even with a well designed FNN architecture, training the FNN efficiently is as challenging as the design of the network. The most popular FNN training algorithm is the well known backpropagation (BP) algorithm, cf. [9]. Although the BP algorithm shares a great convenience of being very simple, early works argue that problems with the BP algorithm is essentially its nature of being a gradient descent algorithm, cf. [10]. Since an overall cost function for training an FNN is often in a large scale and highly non-convex, the BP algorithm suffers from two major drawbacks, namely, (i) existence of undesired local minima, and (ii) slow convergence speed. One major instrument in such a study is via an error/loss surface analysis, cf. [11, 12, 13]. However, even for a very simple problem, such as the classic XOR problem, the error surface analysis

is very complicated and the results are still hard to conclude. Specifically, works in [14, 15, 12, 16] demonstrate such a difficulty of characterising the error surface of an FNN. On the other hand, although the BP algorithm is suspected to be sensitive to initialisations, e.g. [17], recent results reported in [18] suggest that modern FNN learning algorithms can overcome the problem of local optima quite conveniently. Such an observation could be explained by the pioneering works in [19, 20, 21], which developed some conditions on the structure of FNNs to eliminate undesired local minima. Unfortunately, these local minima free conditions seem to be still limited for an application in analysing DNNs.

In order to deal with slow convergence speed of the BP algorithm, various modified BP algorithms have been developed, such as momentum based BP algorithm [22], conjugate gradient algorithm [23], and BFGS algorithm [24]. Although Newton’s method is always of great interest for training an FNN, a complete implementation of Newton’s FNN algorithm is often computationally prohibitive. Instead, certain approximations of the Hessian matrix have been already proposed in the community to deal with the issue, such as a diagonal approximation structure [25], or a block diagonal approximation structure [26]. However, without a true evaluation of the Hessian, performance of these heuristic approximations is hardly convincing. Although existing works [27, 28] have characterised the Hessian matrix by applying the partial derivatives, unfortunately these results fail to provide further information of the Hessian matrix, due to the limitations of partial derivatives. Therefore, in this work, we propose to employ the techniques from multivariable differential calculus to analyse FNNs from a perspective of smooth optimisation.

The paper is organised as follows. In Section 2, by revisiting the classic BP algorithm, we introduce the concepts and techniques from multivariable differential calculus that are crucial in our analysis and development. In Section 3, we characterise the critical point conditions of the general FNN learning cost function, and derive several conditions or design principles for eliminating local minima. With a further analysis of the Hessian matrix of the cost function at the global minima, we characterise the isolatedness properties of the global minima in Section 4. An approximate Newton’s algorithm is developed by leveraging a simple structure of the true Hessian at the global minima. In Section 5, we study the classic XOR problem. Some classic results on XOR networks are reviewed in the developed framework of this paper. Section 6 studies a challenging benchmark of the four-region classification problem with numerical experiments. Finally, conclusions and outlooks are given in Section 7.

2 Revisiting the backpropagation algorithm

By revisiting the classic BP algorithm in this section, we aim to introduce the mathematical notations and concepts used in our development and analysis. We refer to [29, 30] for further readings on the technique.

Let us denote by L the number of layers in an FNN structure, and by n_l the number of processing units in the l -th layer with $l = 1, \dots, L$. Specifically, by letting $l = 0$, we refer to the input layer. Let $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ be an elementary activation function, which is often constructed to be non-constant, continuous, and monotonically increasing. Popular choices of activation function include the Sigmoid function or tanh function. We denote by $\sigma': \mathbb{R} \rightarrow \mathbb{R}$ and $\sigma'': \mathbb{R} \rightarrow \mathbb{R}$ the first derivative and second derivative of the activation function.

For the (l, k) -th unit in an FNN architecture, referring to the k -th unit in the l -th

layer, we can define each unit mapping as

$$\begin{aligned} f_{l,k} : \mathbb{R}^{n_{l-1}} \times \mathbb{R}^{n_{l-1}} \times \mathbb{R} &\rightarrow \mathbb{R}, \\ (w_{l,k}, \phi_{l-1}, b_{l,k}) &\mapsto \sigma(w_{l,k}^\top \phi_{l-1} - b_{l,k}), \end{aligned} \quad (1)$$

where $\phi_{l-1} \in \mathbb{R}^{n_{l-1}}$ denotes the output from the $(l-1)$ -th layer, $w_{l,k} \in \mathbb{R}^{n_{l-1}}$ is the weight vector associated with the (l, k) -th unit, and $b_{l,k} \in \mathbb{R}$ is a scalar bias associated with each unit. For the sake of convenience in technical representation of our analysis, we drop the scalar bias $b_{l,k}$ in our further development. Straightforwardly, we can define the l -th layer evaluation mapping by stacking all unit mappings in the layer as

$$\begin{aligned} F_l : \mathbb{R}^{n_{l-1} \times n_l} \times \mathbb{R}^{n_{l-1}} &\rightarrow \mathbb{R}^{n_l}, \\ (W_l, \phi_{l-1}) &\mapsto [f_{l,1}(w_{l,1}, \phi_{l-1}), \dots, f_{l,n_l}(w_{l,n_l}, \phi_{l-1})]^\top, \end{aligned} \quad (2)$$

with $W_l := [w_{l,1}, \dots, w_{l,n_l}] \in \mathbb{R}^{n_{l-1} \times n_l}$ being the l -th weight matrix. Specifically, let us denote by $\phi_0 \in \mathbb{R}^{n_0}$ the input, then we define $\phi_l := F_l(W_l, \phi_{l-1})$ iteratively.

By composing all the layer-wise mappings together, we end up with the overall network mapping as

$$\begin{aligned} F : \mathbb{R}^{n_{l-1} \times n_l} \times \mathbb{R}^{n_0} &\rightarrow \mathbb{R}^{n_L}, \\ (\mathcal{W}, \phi_0) &\mapsto F_L(W_L, \cdot) \circ \dots \circ F_2(W_2, \cdot) \circ F_1(W_1, \phi_0), \end{aligned} \quad (3)$$

where $\mathcal{W} := (W_1, \dots, W_L) \in \mathbb{R}^{n_0 \times n_1} \times \dots \times \mathbb{R}^{n_{L-1} \times n_L}$. For a specific learning task, one often deploys a suitable error function $E : \mathbb{R}^{n_L} \rightarrow \mathbb{R}$. In this work, we only consider the supervised learning setting with a dataset with T samples, denoted by $(x_i, y_i)_{i=1}^T$. Finally, for a given input $\phi_0 = x_i \in \mathbb{R}^{n_0}$, we define the overall *FNN learning cost function* as

$$\begin{aligned} J : \mathbb{R}^{n_0 \times n_1} \times \dots \times \mathbb{R}^{n_{L-1} \times n_L} \times \mathbb{R}^{n_0} &\rightarrow \mathbb{R}, \\ (\mathcal{W}, \phi_0) &\mapsto (E \circ F)(\mathcal{W}, \phi_0). \end{aligned} \quad (4)$$

It is trivial to see that, if the error function E is differentiable in ϕ_L , then the cost function J is differentiable in both FNN weights \mathcal{W} and the input data ϕ_0 . For the supervised learning setting, i.e., given dataset $(x_i, y_i)_{i=1}^T$, the overall FNN learning cost function can be formed as

$$\mathcal{J} : \mathbb{R}^{n_0 \times n_1} \times \dots \times \mathbb{R}^{n_{L-1} \times n_L} \rightarrow \mathbb{R}, \quad \mathcal{J}(\mathcal{W}) := \sum_{i=0}^T J(\mathcal{W}, x_i). \quad (5)$$

For the convenience of presentations in the rest of the paper, we will drop the summation, only focus on the analysis of sampled cost function J .

In order to develop a gradient descent algorithm to minimise the cost function J . We need to define the derivative of all processing units, i.e., for a given (l, k) -th unit, we have

$$\begin{aligned} f'_{l,k} : \mathbb{R}^{n_{l-1}} \times \mathbb{R}^{n_{l-1}} &\rightarrow \mathbb{R}, \\ (w_{l,k}, \phi_{l-1}) &\mapsto \sigma'(w_{l,k}^\top \phi_{l-1} - b_{l,k}), \end{aligned} \quad (6)$$

and similarly, for the l -th layer

$$\begin{aligned} F'_l : \mathbb{R}^{n_{l-1} \times n_l} \times \mathbb{R}^{n_{l-1}} &\rightarrow \mathbb{R}^{n_l}, \\ (W_l, \phi_{l-1}) &\mapsto [f'_{l,1}(w_{l,1}, \phi_{l-1}), \dots, f'_{l,n_l}(w_{l,n_l}, \phi_{l-1})]^\top. \end{aligned} \quad (7)$$

Algorithm 1: The Backpropagation Algorithm (Batch Learning).

Input : Samples $(x_i, y_i)_{i=1}^T$, an FNN architecture F_{n_1, \dots, n_L} , and initial weights \mathcal{W} ;

Output: Accumulation point \mathcal{W}^* ;

Step 1: For $i = 1, \dots, T$, feed samples x_i through the FNN to compute $\phi_l^{(i)}$ and $\phi_l^{\prime(i)}$ for $l = 1, \dots, L$;

Step 2: Compute the gradient at the error layer
 $\omega_L^{(i)} := \text{diag}(\phi_L^{\prime(i)}) \nabla E(\phi_L^{(i)}) \in \mathbb{R}^{n_L}$;

Step 3: For $l = L, \dots, 1$, compute

$$(1) W_l \leftarrow W_l - \alpha \sum_{i=1}^T \phi_{l-1}^{(i)} (\omega_L^{(i)})^\top;$$

$$(2) \omega_{l-1}^{(i)} \leftarrow \text{diag}(\phi_l^{\prime(i)}) W_l \omega_l^{(i)}, \forall i = 1, \dots, T;$$

Step 4: Repeat from Step 1 until convergence;

For simplicity, we denote $\phi_l' := F_l'(W_l, \phi_{l-1}) \in \mathbb{R}^{n_l}$.

Due to its layer-wise structure of the FNN, by fixing a parameter W_l , we can define a truncated learning cost function, referred to as the l -th *tail error function*, as

$$E_l := E \circ F_L(W_L, \cdot) \circ \dots \circ F_l(W_l, \phi_{l-1}). \quad (8)$$

By applying the chain rule of multivariable derivative, we compute the first derivation of J with respect to W_l in direction $H_l \in \mathbb{R}^{n_{l-1} \times n_l}$ as

$$\begin{aligned} DJ(W_l)H_l &= DE(\phi_L) \cdot D_2 F_L(W_L, \phi_{L-1}) \cdot \dots \cdot \\ &\quad \cdot D_2 F_{l+1}(W_{l+1}, \phi_l) \cdot D_1 F_l(W_l, \phi_{l-1}) H_l, \end{aligned} \quad (9)$$

where $D_1 F_l(W_l, \phi_{l-1})$ and $D_2 F_l(W_l, \phi_{l-1})$ refer to the derivative of F_l with respect to the first and second argument. Explicitly, the first derivative of F_l , i.e., $D_1 F_l(W_l, \phi_{l-1}) : \mathbb{R}^{n_{l-1} \times n_l} \rightarrow \mathbb{R}^{n_l}$ in direction $H_l \in \mathbb{R}^{n_{l-1} \times n_l}$, is computed as

$$D_1 F_l(W_l, \phi_{l-1}) H_l = \text{diag}(\phi_l') H_l^\top \phi_{l-1}, \quad (10)$$

where the operator $\text{diag}(\cdot)$ puts a vector into a diagonal matrix form, and the derivative of F_l with respect to the second parameter in direction $h_{l-1} \in \mathbb{R}^{n_{l-1}}$ as

$$D_2 F_l(W_l, \phi_{l-1}) h_{l-1} = \text{diag}(\phi_l') W_l^\top h_{l-1}. \quad (11)$$

Let us denote $\Sigma_l' := \text{diag}(\phi_l')$ for all $l = 1, \dots, L$. Then, it is straightforward to derive the gradient of J with respect the l -th weight matrix $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ as

$$\nabla J(W_l) = \phi_{l-1} \left(\underbrace{\Sigma_l' W_{l+1} \dots \Sigma_{L-1}' W_L \Sigma_L' \nabla E(\phi_L)}_{=: \omega_l \in \mathbb{R}^{n_l}} \right)^\top, \quad (12)$$

which is a rank-one matrix update. By exploring the layer-wise structure of the FNN, the corresponding vector ω_l , referred to as the l -th *error vector*, can be computed iteratively backwards from the output layer L , i.e.

$$\omega_l := \Sigma_l' W_{l+1} \omega_{l+1}, \quad \text{for all } l = L-1, \dots, 1, \quad (13)$$

with $\omega_L = \Sigma'_L \nabla_E(\phi_L)$. With such a backward mechanism in computing the gradient $\nabla_J(W_l)$, we recover the classic BP algorithm, as presented in Algorithm 1. Such a strategy can be easily adapted to both the batch learning and the online learning setting. In what follows, we will focus on an analysis in the batch learning setting.

3 The problem of local minima: Critical point analysis

With the gradient computed explicitly as in (12), the critical points of the FNN learning cost J are characterised by simply setting it to zero, namely, $\nabla_J(\mathcal{W}) = 0$. More explicitly, for a given sample (x_i, y_i) , we have the following equation system in $\mathcal{W} := (W_1, \dots, W_L) \in \mathbb{R}^{n_0 \times n_1} \times \dots \times \mathbb{R}^{n_{L-1} \times n_L}$ as

$$\begin{cases} \nabla_J^{(i)}(W_L) &= \phi_{L-1}^{(i)} \omega_L^{(i)\top} = 0 \in \mathbb{R}^{n_{L-1} \times n_L} \\ \vdots &= \vdots \\ \nabla_J^{(i)}(W_1) &= \phi_0^{(i)} \omega_1^{(i)\top} = 0 \in \mathbb{R}^{n_0 \times n_1}. \end{cases} \quad (14)$$

Obviously, it is hardly possible to completely characterise all critical points of J . However, the major goal for training an FNN is to minimise the error function $E(\phi_L)$. In other words, it is more sensible to characterise the gradient of the error function E with respect to the output of the FNN, i.e., $\nabla_E(\phi_L)$, instead of the gradient of the overall FNN learning cost $\nabla_J(\mathcal{W})$. In order to guarantee uniqueness of optimal solutions in minimising the error function $E(\phi_L)$, it is reasonable to ensure that the function E is strictly convex. With such a construction, training an FNN has no problems in ending at local minima of the error function.

Principle 1 (Choice of error function). *The error function $E: \mathbb{R}^{n_L} \rightarrow \mathbb{R}$ needs to be strictly convex.*

Now, we rewrite the critical point conditions of the overall learning cost J in Eq. (14), in terms of $\nabla E(\phi_L)$. Similar to the iterative construction of the error vector ω_l as in Eq. (13), we construct a sequence of matrices as, for all $l = L - 1, \dots, 1$,

$$\Psi_l := \Sigma'_l W_{l+1} \Psi_{l+1} \in \mathbb{R}^{n_l \times n_L}, \quad (15)$$

with $\Psi_L = \Sigma'_L \in \mathbb{R}^{n_L \times n_L}$. For a given i -th sample (x_i, y_i) , the equation system as in Eq. (14) can be rewritten as

$$\begin{bmatrix} \Psi_L^{(i)} \otimes \phi_{L-1}^{(i)} \\ \vdots \\ \Psi_1^{(i)} \otimes \phi_0^{(i)} \end{bmatrix} \nabla_E(\phi_L^{(i)}) = 0, \quad (16)$$

where \otimes denotes the Kronecker product of matrices. Let us denote the total number of variables in an FNN by

$$N_{net} = \sum_{l=1}^L n_{l-1} \cdot n_l. \quad (17)$$

Then, by collecting all equation systems from all the samples $i = 1, \dots, T$, we end up

with the following gigantic equation system in $\nabla_E(\phi_L^{(i)})$ as

$$\underbrace{\begin{bmatrix} \Psi_L^{(1)} \otimes \phi_{L-1}^{(1)} & \dots & \Psi_L^{(T)} \otimes \phi_{L-l}^{(T)} \\ \vdots & \ddots & \vdots \\ \Psi_1^{(1)} \otimes \phi_0^{(1)} & \dots & \Psi_1^{(T)} \otimes \phi_0^{(T)} \end{bmatrix}}_{=: \mathbf{P} \in \mathbb{R}^{N_{net} \times (T \cdot n_L)}} \begin{bmatrix} \nabla_E(\phi_L^{(1)}) \\ \vdots \\ \nabla_E(\phi_L^{(T)}) \end{bmatrix} = 0. \quad (18)$$

Obviously, the trivial solution of $\nabla_E(\phi_L^{(i)}) = 0$ for all $i = 1, \dots, T$ is the only solution, if and only if the rank of matrix \mathbf{P} is equal to $T \cdot n_L$. Thus, we just proved the following theorem.

Theorem 1 (Local minima free condition). *Let the error function $E: \mathbb{R}^{n_L} \rightarrow \mathbb{R}$ be strictly convex. Then the FNN learning cost function $J := E \circ F$ is free of local minima, if and only if,*

$$\text{rank}(\mathbf{P}) = T \cdot n_L. \quad (19)$$

Given the number of rows of \mathbf{P} being N_{net} , we conclude our second principle of FNN design as follows.

Principle 2 (Choice of the number of NN variables). *The total number of variables in an FNN, i.e., N_{net} , needs to be greater than or equal to $T \cdot n_L$.*

Now, let us have a closer look at the matrix \mathbf{P} to identify potential strategies to ensure the rank condition to be satisfied as much as possible. By collecting the entries of $\Psi_l^{(i)}$'s and $\phi_l^{(i)}$'s, we construct the following two partitioned matrices

$$\mathbf{\Psi} := \begin{bmatrix} \Psi_L^{(1)} & \dots & \Psi_L^{(T)} \\ \vdots & \ddots & \vdots \\ \Psi_1^{(1)} & \dots & \Psi_1^{(T)} \end{bmatrix} \quad (20)$$

and

$$\mathbf{\Phi} := \begin{bmatrix} \phi_{L-1}^{(1)} & \dots & \phi_{L-l}^{(T)} \\ \vdots & \ddots & \vdots \\ \phi_0^{(1)} & \dots & \phi_0^{(T)} \end{bmatrix}. \quad (21)$$

The two matrices certainly share the same partition structure by construction. Then the matrix \mathbf{P} is simply the Tracy-Singh product of $\mathbf{\Psi}$ and $\mathbf{\Phi}$ as defined in Eq. (18), i.e., the pairwise Kronecker product for each pair of partitions in $\mathbf{\Psi}$ and $\mathbf{\Phi}$, denoted by

$$\mathbf{P} = \mathbf{\Psi} \odot \mathbf{\Phi}. \quad (22)$$

By the properties of the Tracy-Singh product [31], it follows directly

$$\text{rank}(\mathbf{P}) = \text{rank}(\mathbf{\Psi}) \cdot \text{rank}(\mathbf{\Phi}). \quad (23)$$

Since the rank of $\mathbf{\Phi}$ is determined by the data and the FNN architecture, from a design perspective, it is therefore more sensible to guarantee the rank of $\mathbf{\Psi}$. Recall the construction of partitions $\Psi_l^{(i)}$ as in Eq. (15), i.e., products of Σ_l' 's and W_l 's, it is thus reasonable to make these matrix terms as of full rank as possible. We can then conclude the following two principles of designing an FNN.

Principle 3 (Constraints on FNN weights). *Weight matrix $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ needs to be of full rank for all $l = 1, \dots, L$.*

Principle 4 (Choice of activation functions). *The derivative of activation function σ needs to be nonzero for all $x \in \mathbb{R}$, i.e., $\sigma'(x) \neq 0$.*

It is trivial to verify that most of popular activation functions, such as Sigmoid, tanh, logistic, SoftPlus, and SoftSign, are monotonically increasing, hence satisfy Principle 4.

In the rest of this section, we consider an FNN architecture with only one hidden layer, i.e., $L = 2$, and derive a classic result in the literature. We prove the following well known, but still arguable, results as shown in [19].

Proposition 1. *Given an FNN architecture with only one hidden layer, including a dummy unit. Assume that the training dataset consists of T unique samples. Then an FNN learning cost function is free of local minima, if the following four conditions are fulfilled:*

- (1) *The error function E is strictly convex;*
- (2) *There are $T - 1$ units in the hidden layer;*
- (3) *With any weight matrix $W_1 \in \mathbb{R}^{n_0 \times (T-1)}$ in the hidden layer having no identical columns, T unique samples produce a basis in the output space of the hidden layer;*
- (4) *The derivatives of activation functions employed in the output layer are nonzero.*

Proof. Without loss of generality, we can bring the dummy hidden unit to be real, which represents the associated scalar bias and receives a constant input of value 1 from the input layer, i.e., $\phi_1 \in \mathbb{R}^T$. Then the critical point conditions at the output layer $L = 2$ reads as

$$\underbrace{\begin{bmatrix} \Sigma_2^{(1)} \otimes \phi_1^{(1)} & \dots & \Sigma_2^{(T)} \otimes \phi_1^{(T)} \end{bmatrix}}_{:= P_2 \in \mathbb{R}^{(T \cdot n_2) \times (T \cdot n_2)}} \begin{bmatrix} \nabla_E(\phi_2^{(1)}) \\ \vdots \\ \nabla_E(\phi_2^{(T)}) \end{bmatrix} = 0, \quad (24)$$

with $T \cdot n_2$ equations, i.e., P_2 is a square matrix. It is straightforward to conclude that condition (4) ensures

$$\text{rank} \left(\begin{bmatrix} \Sigma_2^{(1)} & \dots & \Sigma_2^{(T)} \end{bmatrix} \right) = n_2. \quad (25)$$

With condition (3), the matrix P_2 is of full rank. Hence, the result follows. \square

Certainly, FNN architectures used in real applications often consist of more hidden layers. Nevertheless, if we fix the l -th hidden layer with $l < L$, then locally the previous $(l - 1)$ -th layer can be treated as the input layer, while the following $(l + 1)$ -th layer can be considered as the output layer. The learning cost function in the output ϕ_{l+1} is nothing but the $(l + 1)$ -th tail error function as defined in Eq. (8). The results in Proposition 1 can be used for choosing the number of units in the hidden layers.

Principle 5 (Choice of the number of hidden units). *For a given l -th hidden layer with $l < L$, we assume that there are T_l unique patterns after forwarding through the previous $l - 1$ layers. Then we need at least $T_l - 1$ units in this layer to ensure a potential exemption of local minima.*

This design principle is very heuristic, since knowledge of the number of patterns in the hidden layers are generally unknown. When pooling is used in a feedforward convolutive neural network, it might be interesting to see the role of this principle. We treat this aspect as one potential research direction in the future.

4 Hessian analysis and approximate Newton's method

The information from the Hessian matrix is critically important for designing efficient numerical algorithms. Specifically, definiteness of the Hessian matrix is the indicator to the isolatedness of the critical points, which will affect significantly the convergence speed of the algorithms. The Hessian form of the FNN learning cost function J is a bilinear operator $H_J: \mathbb{R}^{N_{net}} \times \mathbb{R}^{N_{net}} \rightarrow \mathbb{R}$, computed by the second derivative of J . Its computation is rather straightforward but very tedious. For the sake of readability, we only present one component of the second derivative with respect to two specific indices k and l , i.e.,

$$\begin{aligned} D^2 J(\mathcal{W})(H_k, H_l) &= \frac{d^2}{dt^2} J_{l,k}(\mathcal{W} + t\mathcal{H}) \Big|_{t=0} \\ &= D^2 E(\phi_L)(DF_{l,L}(W_l)H_l, DF_{k,L}(W_k)H_k) \\ &\quad + \sum_{\substack{i=1 \\ i < l, k}}^L (W_i^\top \dots \Sigma'_k H_k^\top \phi_{k-1})^\top \cdot \text{diag}(\nabla_J(W_i)) \cdot \Sigma''_i \cdot (W_i^\top \dots \Sigma'_l H_l^\top \phi_{l-1}), \end{aligned} \quad (26)$$

where $\Sigma''_i \in \mathbb{R}^{n_i \times n_i}$ is the diagonal matrix with the second derivative of the activation functions on the diagonal. Let ϕ_L^* be the desired output of the FNN, reached by an global minima \mathcal{W}^* . Then all gradients $\nabla_E(\phi_L^*)$ of the error function are equal to zero. As a sequel, the second summand in the last equation in Eq. (26) vanishes. Therefore, the Hessian $H_J(\mathcal{W})$ evaluated at \mathcal{W}^* is given as

$$\begin{aligned} D^2 J(\mathcal{W}^*)(H_k, H_l) &= \frac{d^2}{dt^2} J_{l,k}(\mathcal{W}^* + t\mathcal{H}) \Big|_{t=0} \\ &= \sum_{l,k=1}^L (DF_{l,L}(W_l^*)H_l)^\top H_E(\phi_L^*) DF_{k,L}(W_k^*)H_k, \end{aligned} \quad (27)$$

where $H_E(\phi_L): \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \rightarrow \mathbb{R}$ is the Hessian form of the error function E . For a given sample (x_i, y_i) , we construct the following two identically partitioned matrices, i.e.,

$$\Psi^{*(i)} := \begin{bmatrix} \Psi_L^{(i)} \\ \vdots \\ \Psi_1^{(i)} \end{bmatrix} H_E(\phi_L^*) \begin{bmatrix} \Psi_L^{(i)} \\ \vdots \\ \Psi_1^{(i)} \end{bmatrix}^\top \quad (28)$$

and

$$\Phi^{*(i)} := \begin{bmatrix} \phi_{L-1}^{*(i)} \\ \vdots \\ \phi_0^{*(i)} \end{bmatrix} \begin{bmatrix} \phi_{L-1}^{*(i)} \\ \vdots \\ \phi_0^{*(i)} \end{bmatrix}^\top. \quad (29)$$

Then, the Hessian of J at a global minima \mathcal{W}^* can be represented in a matrix form as

$$H_J(\mathcal{W}^*) = \sum_{i=1}^T \Psi^{*(i)} \odot \Phi^{*(i)} \in \mathbb{R}^{N_{net} \times N_{net}}. \quad (30)$$

If the activation functions used at the output layer follow Principle 4, then the rank of matrix $\Phi^{*(i)}$ as defined in Eq. (28) is determined by the rank of the Hessian $H_E(\phi_L^{*(i)})$. If the error function E is chosen according to Principle 1, then it needs to be further assume have a non-degenerate Hessian, i.e., E being a Morse function, cf. [32] Consequently, we have

$$\text{rank}(\Psi^{*(i)}) = n_L. \quad (31)$$

Hence, we introduce the next design principle on the choice of error function, in addition to Principle 1.

Principle 1.a (Strong choice of error function). *The error function $E: \mathbb{R}^{n_L} \rightarrow \mathbb{R}$ needs to be strictly convex and Morse, i.e., the Hessian $H_E(\phi_L)$ is non-degenerate for all $\phi_L \in \mathbb{R}^{n_L}$.*

Since both matrices $\Psi^{*(i)}$ and $\Phi^{*(i)}$ are positive semi-definite, the Hessian matrix $H_J(\mathcal{W}^*)$ is simply a sum of T rank- n_L positive semi-definite matrices. We can conclude the following result.

Theorem 2. *Given an FNN architecture satisfying Principle 1, 2, 4, 1.a, and a training dataset consisting of T unique samples. If a global minimum \mathcal{W}^* of the FNN learning cost is reachable, then the rank of the Hessian matrix of J is bounded from above by*

$$\text{rank}(H_J(\mathcal{W}^*)) \leq T \cdot n_L. \quad (32)$$

It is important to notice that the Hessian $H_J(\mathcal{W}^*)$ is neither diagonal nor block diagonal, which demotivates the existing approximate strategies of the Hessian in [25, 26]. With our explicit characterisation of the Hessian at global minima, we propose to approximate the Hessian of J at arbitrary point \mathcal{W} with the structure as shown in Eq. (30). The corresponding approximate Newton's algorithm is presented in Algorithm 2. It is important to notice that, in Step 6, a scaled identity matrix with $\delta > 0$ is added to the approximate Hessian in order to ensure the existence of the inverse. Moreover, the computation of the Newton update in Step 6 can be efficiently carried out by implementing a block Cholesky decomposition.

5 Case study: The XOR problem

The XOR problem is a classic and most well known demonstrative scenario for FNN training. Two specific network structures, i.e., the 2-2-1 XOR network and the 2-3-1 XOR network, have been extensively studied in the literature. In this section, we apply our developed analysis to investigate some classic results on the XOR networks.

We confine ourselves to the network structure with only one hidden layer, while the structure of the input and output layers are fixed. Namely, we have $L = 2$, $n_0 = 2$, and $n_2 = 1$. All processing units are chosen to be the Sigmoid function. We define the input patterns as

$$X := [x_1, x_2, x_3, x_4] = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}, \quad (33)$$

and correspondingly, the output

$$Y := [y_1, y_2, y_3, y_4] = [0, 1, 1, 0] \in \mathbb{R}^{1 \times 4}. \quad (34)$$

Algorithm 2: An Approximate Newton's Backpropagation Algorithm (Batch Learning).

Input : Samples $(x_i, y_i)_{i=1}^T$, an FNN architecture F_{n_1, \dots, n_L} , and initial weights $\mathcal{W} \in \mathbb{R}^{n_0 \times n_1} \times \dots \times \mathbb{R}^{n_{L-1} \times n_L}$;

Output: Accumulation point \mathcal{W}^* ;

Step 1: For $i = 1, \dots, T$, feed samples x_i through the FNN to compute $\phi_l^{(i)}$ and $\phi'_l{}^{(i)}$ for $l = 1, \dots, L$;

Step 2: Compute

- (1) $\Psi_L^{(i)} := \text{diag}(\phi'_L{}^{(i)}) \in \mathbb{R}^{n_L \times n_L}$;
- (2) $\omega_L^{(i)} := \text{diag}(\phi'_L{}^{(i)}) \nabla_E(\phi_L^{(i)}) \in \mathbb{R}^{n_L}$;
- (3) $H_E(\phi_L^{(i)}) \in \mathbb{R}^{n_L \times n_L}$;

Step 3: For $l = L, \dots, 1$ and $i = 1, \dots, T$, compute

- (1) $\Psi_{l-1}^{(i)} := \text{diag}(\phi'_{l-1}{}^{(i)}) W_l \Psi_l^{(i)} \in \mathbb{R}^{n_{l-1} \times n_{l-1}}$;
- (2) $\omega_{l-1}^{(i)} \leftarrow \Psi_{l-1}^{(i)} \nabla_E(\phi_L^{(i)}) \in \mathbb{R}^{n_{l-1}}$;

Step 4: For $l = L, \dots, 1$, compute the gradient

$$\nabla_J(W_l) = \sum_{i=1}^T \phi_{l-1}^{(i)} (\omega_l^{(i)})^\top ;$$

Step 5: Compute the approximate Hessian $H_J(\mathcal{W})$ according to Eq. (30) ;

Step 6: Compute the Newton update

$$\mathcal{W} \leftarrow \mathcal{W} - \alpha (H_J(\mathcal{W}) + \delta I_{N_{net}})^{-1} \nabla_J(\mathcal{W}) ;$$

Step 7: Repeat from Step 1 until convergence;

There are $T = 4$ unique samples. Then the FNN learning cost function is defined as

$$J_{XOR}(\mathcal{W}) = \frac{1}{2} \sum_{i=1}^4 (F(\mathcal{W}, x_i) - y_i)^2 \quad (35)$$

where $\mathcal{W} := \{W_1, W_2\}$ with $W_1 \in \mathbb{R}^{2 \times n_1}$ and $W_2 \in \mathbb{R}^{n_1 \times 1}$.

Firstly, let us have a look at the 2-2-1 XOR network, i.e., $n_1 = 2$. We then investigate the rank of the following matrix

$$\mathbf{P}_{2,2,1} := \begin{bmatrix} \Psi_2^{(1)} \otimes \phi_1^{(1)} & \dots & \Psi_2^{(4)} \otimes \phi_1^{(4)} \\ \Psi_1^{(1)} \otimes \phi_0^{(1)} & \dots & \Psi_1^{(4)} \otimes \phi_0^{(4)} \end{bmatrix}. \quad (36)$$

By considering the biases associated with each unit, the total number of FNN variables is

$$N_{2,2,1} = 3 \cdot 2 + 3 \cdot 1 = 9. \quad (37)$$

Since $n_2 = 1$, it is easily to see that

$$N_{2,2,1} \geq T \cdot n_2 = 4. \quad (38)$$

We can construct two matrices

$$\Psi^{*(i)} := \begin{bmatrix} \Psi_2^{(i)} \\ \Psi_1^{(i)} \end{bmatrix} H_E(\phi_2^{*(i)}) \begin{bmatrix} \Psi_2^{(i)} \\ \Psi_1^{(i)} \end{bmatrix}^\top \quad (39)$$

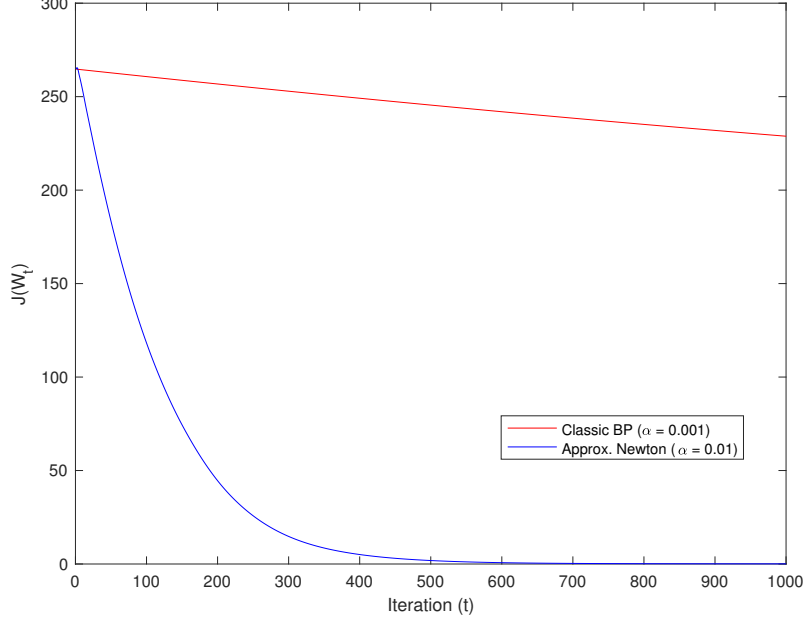


Figure 1: Comparison of convergence speed.

and

$$\Phi^{*(i)} := \begin{bmatrix} \phi_1^{*(i)} \\ \phi_0^{*(i)} \end{bmatrix} \begin{bmatrix} \phi_1^{*(i)} \\ \phi_0^{*(i)} \end{bmatrix}^\top. \quad (40)$$

Since the Hessian $H_E(\phi_2)$ is a scalar, it is trivial to see that the Hessian of J_{XOR} is a sum of four rank-one matrices, i.e., $\text{rank}(H_J)(\mathcal{W}) \leq 4$, hence degenerate. We then conclude the classic results as in [15, 11].

Corollary 1. *Let a 2-2-1 XOR network architecture satisfy all four Principles 1-4. Then the network has no local minima and global minima have a degenerate Hessian.*

Similar analysis on the 2-3-1 XOR network leads to the following results.

Corollary 2. *Let a 2-3-1 XOR network architecture satisfy three Principles 1, 2, 4. Then the network has no local minima and global minima have a degenerate Hessian.*

Remark 1. *It is clear then that any XOR network with an architecture 2- n_1 -1 for $n_1 \geq 2$ has only degenerate global minima.*

6 Numerical experiments

In our numerical experiments, we investigate performance of our proposed approximate Newton's algorithm on the four regions classification benchmark, as originally proposed in [33]. In \mathbb{R}^2 around the origin, we have a square area $(-4, 4) \times (-4, 4)$, and three concentric circles with their radiuses being 1, 2, and 3. Four regions/classes

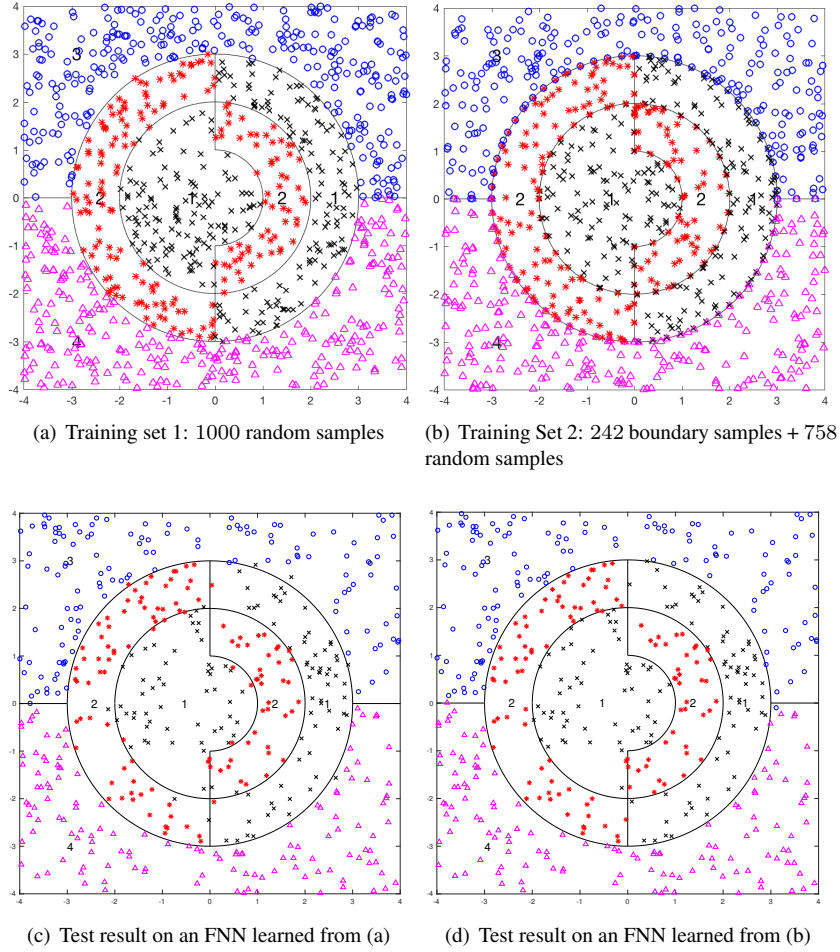


Figure 2: Illustration of Learning Outcome in solving the four regions classification problem.

are interlocked, nonconvex, as shown in Figure 2(a). We draw randomly $T = 1000$ samples in the box for training, and specify the corresponding output to be the i -th basis vector in \mathbb{R}^4 . We deploy an FNN architecture with two hidden layers, i.e., $L = 3$. In both hidden layer, there are 10 units each. Hence, we have $n_0 = 2$, $n_1 = n_2 = 10$, and $n_3 = 4$. All activation functions are chosen to be Sigmoid. Finally, the standard least squares error is used as the error function, which is Morse and strictly convex.

Our experiments indicates that the approximate Newton's algorithm is capable of handling vanishing gradient, and reach much lower value in the error function. More specifically, the convergence speed of the proposed approximate Newton's FNN algorithm is much faster than the classic BP algorithm, shown in Figure 1.

By random sampling, the learned FNN produces quite accurate results as shown in Figure 2(c), except the points on close to the boundary. In order to copy with such an issue, we construct a framed dataset, consisting 242 samples on the two sides along the boundary with the other 758 random samples shown in Figure 2(b). We test the same test sample with the FNN learned from the framed dataset, leading to a much more

accurate classification shown in Figure 2(d).

7 Conclusion

In this work, we developed a smooth optimisation perspective on the challenge of designing and training an FNN architecture in the supervised learning setting. By characterising the critical point conditions of the overall learning cost function, we derive some mild conditions to ensure an FNN to be free of local minima. Classic results on this matter in both XOR scenario and general setting are also reviewed. Our analysis also identifies the Hessian structure of the cost function at the global minima, which leads to an approximate Newton’s FNN algorithm.

References

- [1] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1996.
- [2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, 2015.
- [3] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*. Springer-Verlag, London, 2015.
- [4] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10)*, vol. 9, 2010, pp. 249–256.
- [5] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [6] S. Sun, W. Chen, L. Wang, X. Liu, and T.-Y. Liu, “On the depth of deep neural networks: A theoretical view,” in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2016, pp. 2066–2072.
- [7] H. N. Mhaskar and C. A. Micchelli, “How to choose an activation function,” in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, 1993, pp. 319–326.
- [8] T. Falas and A. G. Stafylopatis, “The impact of the error function selection in neural network-based classifiers,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, vol. 3, 1999, pp. 1799–1804.
- [9] B. Widrow and M. A. Lehr, “30 years of adaptive neural networks: perceptron, madaline, and backpropagation,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.
- [10] R. S. Sutton, “Two problems with backpropagation and other steepest-descent learning procedures for networks,” in *Proceedings of the 8-th Annual Conference of the Cognitive Science Society*, 1986, pp. 823–831.
- [11] L. G. C. Hamey, “XOR has no local minima: A case study in neural network error surface analysis,” *Neural Networks*, vol. 11, no. 4, pp. 669–681, 1998.

- [12] I. G. Sprinkhuizen-Kuyper and E. J. W. Boers, "The local minima of the error surface of the 2-2-1 xor network," *Annals of Mathematics and Artificial Intelligence*, vol. 25, no. 1, pp. 107–136, 1999.
- [13] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015, pp. 192–204.
- [14] P. J. G. Lisboa and S. J. Perantonis, "Complete solution of the local minima in the xor problem," *Network: Computation in Neural Systems*, vol. 2, no. 1, pp. 119–124, 1991.
- [15] I. G. Sprinkhuizen-Kuyper and E. J. W. Boers, "The error surface of the simplest xor network has only global minima," *Neural Computation*, vol. 8, no. 6, pp. 1301–1320, 1996.
- [16] —, "A local minimum for the 2-3-1 xor network," *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 968–971, 1999.
- [17] J. F. Kolen and J. B. Pollack, "Backpropagation is sensitive to initial conditions," *Complex Systems*, vol. 4, no. 3, pp. 269–280, 1990.
- [18] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems," Published at the 5th International Conference on Learning Representations (ICLR). arXiv:1412.6544., 2015.
- [19] X.-H. Yu, "Can backpropagation error surface not have local minima," *IEEE Transactions on Neural Networks*, vol. 3, no. 6, pp. 1019–1021, 1992.
- [20] X.-H. Yu and G.-A. Chen, "On the local minima free condition of backpropagation learning," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1300–1303, 1995.
- [21] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 1, pp. 76–86, 1992.
- [22] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the convergence of the back-propagation method," *Biological Cybernetics*, vol. 59, no. 4, pp. 257–263, 1988.
- [23] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," *IEE Proceedings G - Circuits, Devices and Systems*, vol. 139, no. 3, pp. 301–310, 1992.
- [24] Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Ng, "On optimization methods for deep learning," *Proceedings of international conference on Machine Learning*, 2011.
- [25] R. Battiti, "First- and second-order methods for learning: Between steepest descent and newton's method," *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992.

- [26] Y.-J. Wang and C.-T. Lin, “A second-order learning algorithm for multilayer networks based on block Hessian matrix,” *Neural Networks*, vol. 11, no. 9, pp. 1607–1622, 1998.
- [27] C. Bishop, “Exact calculation of the hessian matrix for the multilayer perceptron,” *Neural Computation*, vol. 4, no. 4, pp. 494–501, 1992.
- [28] E. Mizutani and S. E. Dreyfus, “Second-order stagewise backpropagation for hessian-matrix analyses and investigation of negative curvature,” *Neural Networks*, vol. 21, no. 2-3, pp. 193–203, 2008.
- [29] H. Amann and J. Escher, *Analysis II*. Birkhäuser Verlag, 2008.
- [30] J. Nocedal and S. J. Wright, *Numerical Optimization, 2nd Ed.* New York: Springer, 2006.
- [31] D. S. Tracy and R. P. Singh, “A new matrix product and its applications in partitioned matrices,” *Statistica Neerlandica*, vol. 26, pp. 143–157, 1972.
- [32] J. Milnor, *Morse Theory*. Princeton University Press, 1963.
- [33] S. Singhal and L. Wu, “Training multilayer perceptrons with the extended Kalman algorithm,” in *Advances in Neural Information Processing Systems*, 1989, pp. 133–140.